Design de Software – CI1316
Documentação do Projeto "Seu Cantinho"

Leonardo Amorim Carpwiski[1]
Pedro Pinheiro de Freitas Filho[1]

[1] Departamento de Informática
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{lac23, ppff23}@inf.ufpr.br

1. Introdução

A escolha da arquitetura é determinante para o sucesso de uma aplicação, impactando diretamente desde a execução em tempo real até a facilidade de implementação de novas features. Trata-se de um exercício de equilíbrio (trade-off): cada decisão arquitetural deve ser guiada pelas necessidades críticas do sistema. Portanto, a seleção de padrões e componentes deve visar não apenas a funcionalidade imediata, mas também garantir resiliência contra falhas, escalabilidade para suportar crescimento de carga e uma confiabilidade que assegure a integridade dos dados e a continuidade do serviço.

Essa documentação trata a respeito do projeto "Seu Cantinho", da disciplina de Design de Software do Curso de Ciência da Computação, Universidade Federal do Paraná. O problema abordado trata a respeito de criar um protótipo para os problemas que Dona Maria está enfrentando em seu sistema, precisando modelar a arquitetura de forma correta, a fim de atender seus requisitos. O repositório do Github, com os fontes de execução do protótipo e os fontes dos diagramas, assim como as instruções para execução do código estão acessíveis neste link.

2. Descrição da Arquitetura Escolhida

A arquitetura escolhida para a modelagem do projeto foi **Microsserviços**, que consiste de serviços delimitados de forma concisa e fracamente acoplados entre si. Dessa forma, os microsserviços podem ser ajustados e executados de forma independente aos outros, facilitando a manutenção do código e garantindo a escalabilidade do sistema. Se um serviço necessita de comunicar ou de dados com outro, ele faz a requisição diretamente ao outro serviço e o responsável por atender a requisição é o serviço chamado. Isso garante a alta coesão e o baixo acoplamento, visto que os serviços agora possuem suas próprias responsabilidades e seus dados.

Foram elaborados um diagrama de classes de um diagrama de componentes para ilustrar a modelagem do sistema de Microsserviços para o problema do "Seu Cantinho". A Figura 1 mostra o diagrama de classes, enquanto que a Figura 2 apresenta o diagrama de componentes.

Pela estrutura do diagrama de classes, é possível notar a presença de uma interface, que lê os dados de um usuário, podendo ser Lojista ou Cliente, e entrega essa requisição a um controlador geral, que irá designar a requisição feita para a

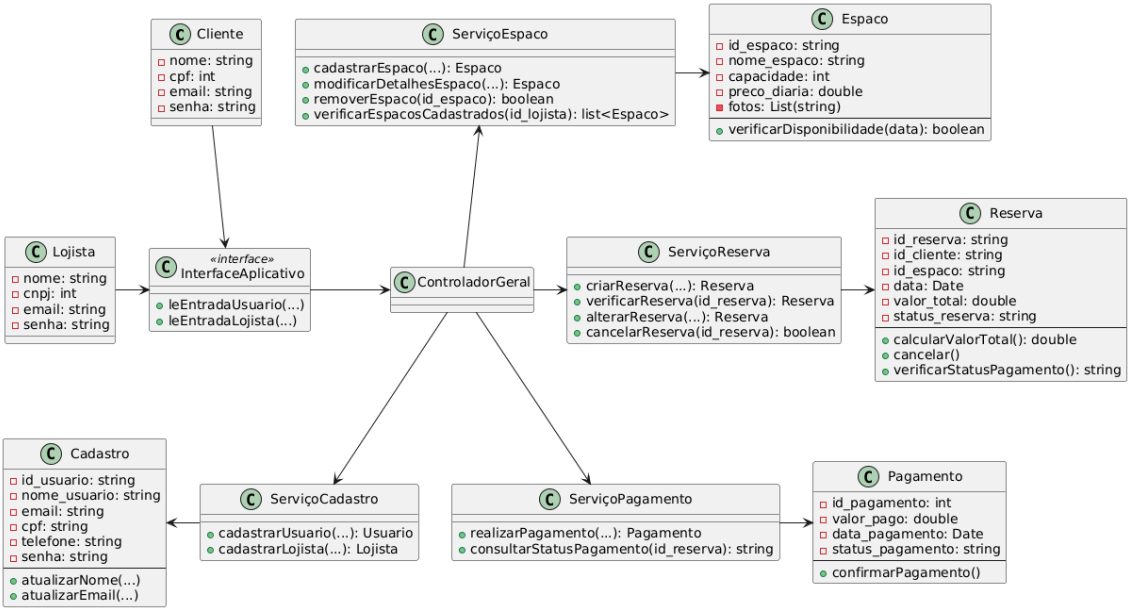

Figura 1. Diagrama de Classes da arquitetura Microsserviços.

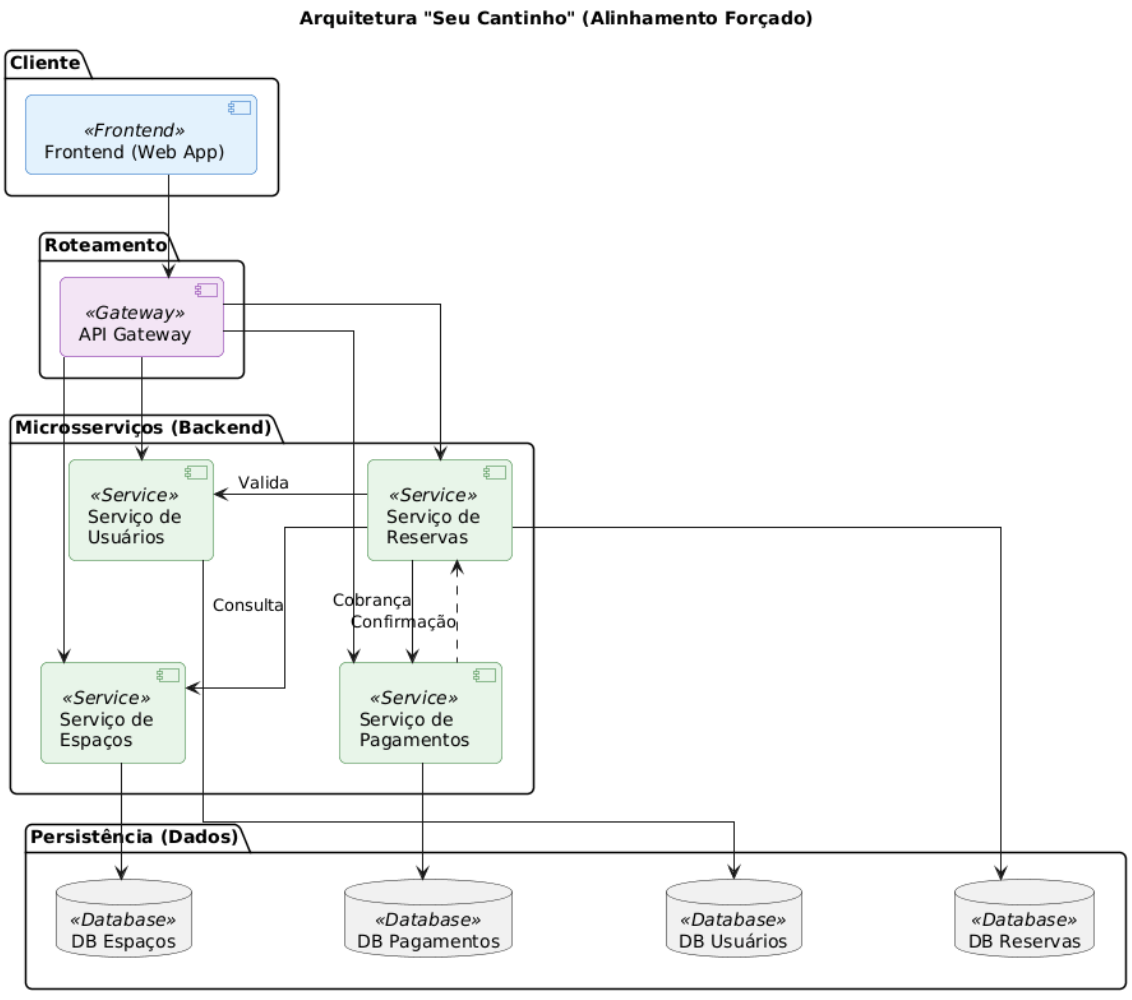

Figura 2. Diagrama de Componentes da arquitetura Microsserviços.

classe do serviço correspondente. Cada serviço usa apenas os dados aos quais possui acesso, de forma coesa e coerente ao propósito da classe. O padrão GRASP tentou ser representado através do controlador (*padrão Controller*), dos serviços e seus dados correspondentes (*padrão Creator*) e das regras estabelecidas nas classes de dados (*padrão Information Expert*).

No diagrama de componentes, é perceptível a correlação entre os serviços, que recebem as requisições do API Gateway (nesse caso, usando as regras da API REST), onde cada um dos serviços se comunica ou não com outros serviços, requisitando dados que somente os serviços podem acessar. Dessa forma, como dito anteriomente, o acoplamento é reduzido ao nível de serviço e a coesão é mantida em alto nível onde cada serviço é responsável por suas próprias tarefas e conjunto de dados.

Com relação ao código efetuado e disponível no repositório, mostra-se que a arquitetura de Microsserviços é refletida na separação de cada serviço em seu respectivo diretório, com os componentes do Docker (`Dockerfile` e `package.json`) e do OpenAPI (`openapi.yaml`), além de manter o conjunto de dados restrito apenas ao próprio serviço no diretório `/database`. Dessa forma, ao criar os diversos contêineres no Docker, os microsserviços são iniciados e efetuam as tarefas correspondentes da API REST (`GET`, `POST`, `PATCH`, `DELETE`, etc). Um serviço, no código, nunca acessa outro serviço diretamente; ele faz uma requisição HTTP que fornece a outro serviço um trabalho a ser realizado, devolvendo os dados correspondentes à requisição, ou um código de erro em falha.

3. Justificativas do Design

Analisando as requisições feitas por Dona Maria, existem as seguintes pendências no sistema de reservas de lojas:

- **Sistema Inicial Simples:** Inicialmente, a modelagem do sistema provavelmente foi desenvolvida com base na arquitetura **Monolítica**, fazendo com que os elementos do sistema estejam fortemente acoplados. Isso reflete uma baixa coesão dos elementos e um alto acoplamento, com uma interferência em uma parte se alastrando por todo o sistema, o que é possível resolver com outras arquiteturas.
- **Consistência entre Filiais:** O problema da consistência dos dados de espaços, reservas e pagamentos é crítico, visto que com a arquitetura Monolítica fica difícil identificar os trechos correspondentes para ajustar o código problemático e os dados ficam centralizados em um só Banco de Dados, sem que um pedaço específico (por exemplo, o código de reserva) mexa apenas no seu respectivo dado.
- **Limitações de Usabilidade e Confiabilidade:** A arquitetura Monolítica com alto índice de acoplamento fornece um sistema complexo de usabilidade ao usuário, onde as funcionalidades de reserva e pagamento podem estar atreladas, ao invés de específicas para cada operação do usuário. Além disso, há também o problema de confiabilidade, visto que se uma parte do sistema cai, todo o sistema cai junto (por exemplo, se não é possível verificar o sistema de reservas, não tem como pagar uma reserva específica).
- **Problema de *Double-booking*:** A complexidade do código fortemente conectado da arquitetura Monolítica atual apresenta sérios problemas de remar-

cação de reservas, justamente pelo fato de que a baixa coesão dos elementos de código traz dificuldades de ajuste e verificação dos dados das reservas. O problema de reservar um espaço já alocado é sério e deve ser ajustado com uma modelagem adequada, com acesso coeso aos dados das reservas.

- **Perda de Informação Financeira:** O sistema atual apresenta uma baixa confiabilidade dos dados financeiros dos clientes, o que é também um problema grave, visto que interfere legalmente com os consumidores da aplicação. Dessa forma, é necessário um sistema que forneça alta confiabilidade dos dados de pagamento.
- **Aumento no Tempo de Operação:** O alto acoplamento dos componentes do sistema faz com que a execução de uma parte dependa diretamente de outra, de forma nada coesa, aumentando de forma desnecessária o tempo de execução do programa. Assim, é necessária uma arquitetura em que os componentes consigam operar de forma independente, não necessariamente assíncronos, para melhor estabilidade e desempenho do sistema.

Dados todos esses problemas identificados no enunciado do trabalho, a respeito do sistema atual que existe para o "Seu Cantinho", é possível identificar os **Requisitos Arquitetonicamente Significantes**, em busca de adequar uma modelagem arquitetural nova para o sistema. São eles:

- **Atributos de Desempenho:**
  - *Latência:* O sistema deve processar a confirmação de uma reserva e retornar o comprovante ao usuário em menos de 2 segundos para 95% das requisições (P95 $<$ 2s).
    Prioridade: Alta
  - *Throughput:* O sistema deve suportar até 100 pedidos de reserva simultâneos por minuto sem degradação do tempo de resposta.
    Prioridade: Média
- **Atributos de Escalabilidade:**
  - *Elasticidade:* Em picos de acesso (ex: datas comemorativas), o serviço de Reservas deve escalar horizontalmente de 2 para até 10 instâncias automaticamente se a CPU exceder 70%.
    Prioridade: Alta
- **Atributos de Disponibilidade:**
  - *Uptime:* O módulo de Reservas deve manter uma disponibilidade de 99,9% durante o horário comercial (08:00 às 22:00), permitindo no máximo 43 minutos de inatividade por mês.
    Prioridade: Alta
  - *Recuperação (RTO):* Em caso de falha crítica no serviço de Pagamentos, o sistema deve ser capaz de reiniciar e restaurar a operação em menos de 5 minutos.
    Prioridade: Média
- **Atributos de Confiabilidade:**
  - *Consistência de Dados:* O sistema deve garantir zero ocorrências de *double-booking* (duas reservas para o mesmo espaço/horário), mesmo sob concorrência alta.
    Prioridade: Alta

  - *Tolerância a Falhas:* Se o serviço de Notificações falhar, a reserva deve ser concluída com sucesso, enfileirando o e-mail para envio posterior (degradação graciosa).
    Prioridade: Alta
- **Atributos de Usabilidade:**
  - *Acessibilidade:* O portal do cliente deve atingir conformidade nível AA da WCAG para garantir acesso a pessoas com deficiência visual.
    Prioridade: Média
  - *Curva de Aprendizado:* Um novo funcionário da Dona Maria deve ser capaz de realizar um cancelamento de reserva em menos de 3 cliques e sem consultar o manual.
    Prioridade: Baixa (Foco atual é backend)
- **Atributos de Segurança:**
  - *Confidencialidade:* Todos os dados de cartão de crédito e dados sensíveis (PII) devem ser criptografados em repouso (AES-256) e em trânsito (TLS 1.3).
    Prioridade: Alta
  - *Integridade/Auditoria:* Todas as transações financeiras devem gerar logs imutáveis para auditoria, garantindo conformidade com a LGPD.
    Prioridade: Alta

Dado os requisitos anteriormente citados, há a justificativa da escolha de arquitetura de **Microsserviços** como comentado na seção anterior, visto que esta abordagem resolve diretamente os problemas de acoplamento e coesão identificados no sistema Monolítico. Ao dividir o sistema em serviços autônomos (Reservas, Pagamentos, Espaços), garantimos a Disponibilidade, pois a falha em um serviço não derruba os demais. Os requisitos de Segurança são cruciais para o devido funcionamento do sistema, devido ao fato de que o sistema manipula dados financeiros de terceiros. Além disso, o sistema de Microsserviços garante tolerância a falhas, visto que a queda de um serviço não necessariamente afeta o funcionamento de outro, e também possibilita o uso de outras instâncias do serviço em diferentes dispositivos. Por fim, a Escalabilidade elástica é facilitada, permitindo que recursos sejam alocados especificamente onde há maior carga, otimizando o desempenho e a confiabilidade financeira exigida por Dona Maria.

4. Reflexões Finais

A análise detalhada das dificuldades enfrentadas por Dona Maria serviu como guia para definir a melhor forma de organizar o novo sistema. Esse processo deixa evidente que a escolha da estrutura do software é uma decisão fundamental para o sucesso do projeto a longo prazo.

Quando se dedica tempo para planejar bem essa base, os benefícios vão muito além de apenas resolver as falhas que existem hoje. Um sistema bem desenhado garante que as informações fiquem seguras e que o funcionamento seja estável no dia a dia. Além disso, essa organização facilita o trabalho futuro: torna muito mais simples realizar ajustes ou melhorias e permite que a tecnologia acompanhe o crescimento do negócio sem a necessidade de refazer todo o trabalho do zero quando novas demandas surgirem.